Conceptos Básicos, TADS, Estructura de Datos, POO

Pablo F. Castro
UNRC

Resumen de la Clase

- Conceptos básicos: Abstracción, tipos de datos, estructura de datos, tipos de datos abstractos.
- Definición de tipos abstractos de datos, un ejemplo.
- Especificación de tipos abstractos de datos, axiomas, pre y postcondiciones.
- Implementación de tipos abstractos de datos, ejemplos.

Abstracción

El proceso de abstracción nos permite olvidarnos de ciertos detalles para concentrarnos en aquellos importantes.

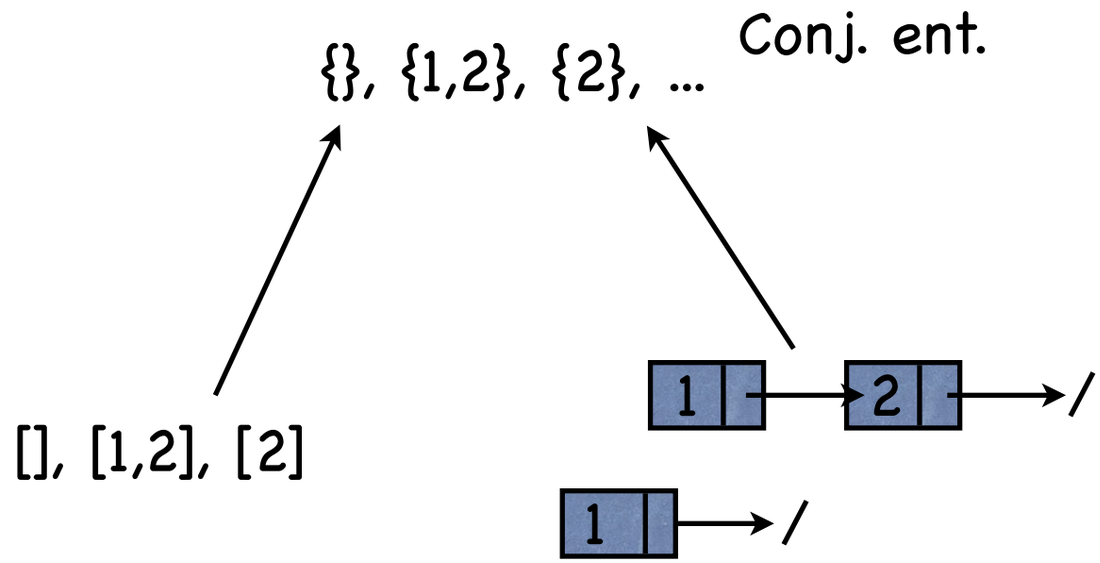
> Si queremos razonar sobre problemas complejos debemos utilizar abstracciones

Abstracción usando procedimientos

```
function f(x,y: integer):result integer;
begin
  f := (x*x) + (y*y)
end
```

(5*5+7*7),(10*10+2*2), ...,(1*1+2*2)

Abstracción de Datos



Conceptos Básicos

Tipos de Datos (o tipo): Conjunto de valores y operaciones sobre ellos (un álgebra).

{0,1,2,3,...}, +, 0, *

Tipos Básicos: Son aquellos provistos por los leng. de programación.

```
var x: Integer
```

Estructuras de Datos: Es una forma de organizar datos para facilitar su manipulación.

```
var a:Array[1..10] of Integer
```

Tipo Abstracto de Datos: Son modelos matemáticos que nos permiten realizar abstracciones sobre datos.

Tipos Abstractos de Datos

Podemos dar la siguiente definición de TAD (B.Liskov):

> Un TAD define una colección de elementos abstractos los cuales son caracterizados mediante un conjunto operaciones sobre estos.

Esta idea esta relacionada con el concepto de Algebra de matemática

Podemos pensar a los TADs como abstracciones de tipos. Las operaciones y los elementos se definen describiendo las propiedades que cumplen, y no explícitamente

Ejemplo: Pilas

El TAD pila especifica una colección de datos que se comporta como una pila de objetos:

Una pila de panqueques.

Cuando ponemos un panqueque nuevo lo ponemos en el tope

Cuando sacamos un panqueque sacamos uno del tope

Los panqueques están ordenados de un forma lineal (unos tras otro).

El TAD Pila

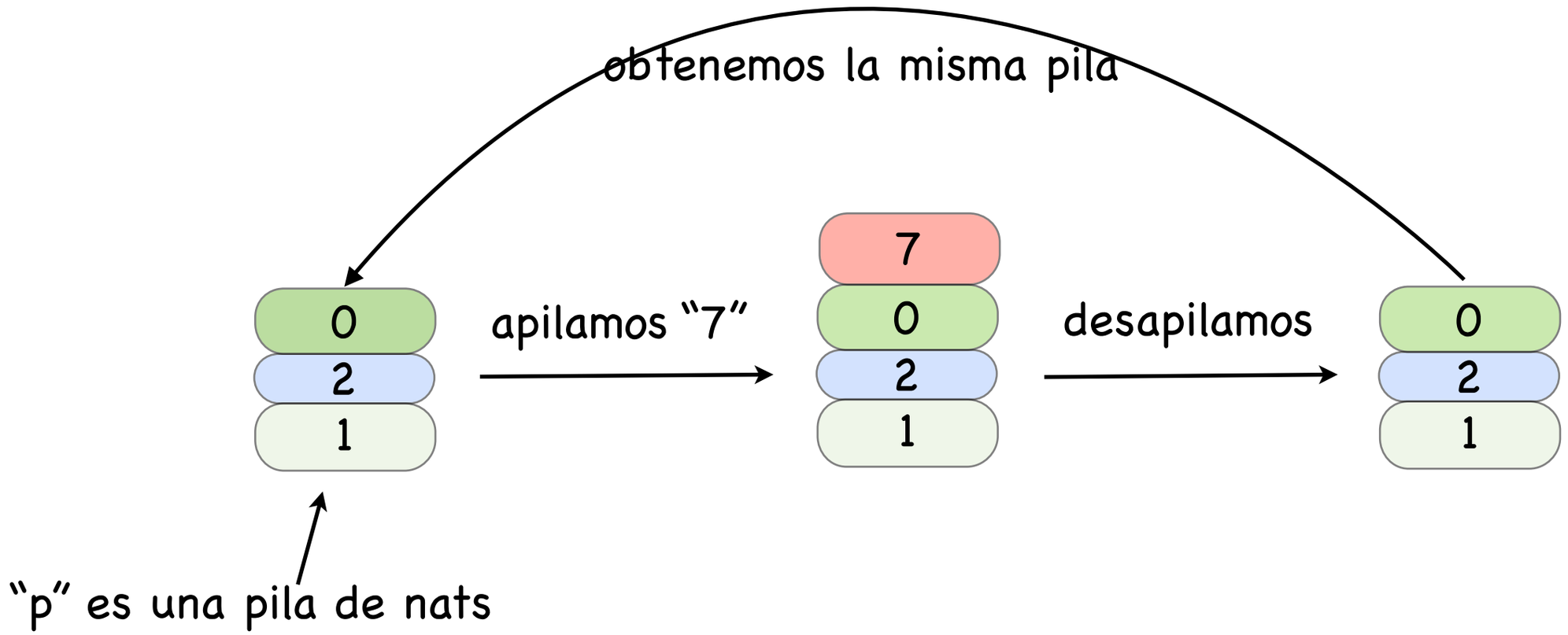
El TAD pila (o stack) se define de la siguiente forma:

- Elementos: secuencia lineal de elementos de un tipo dado, alguno de los extremos es llamado tope
- Operaciones:

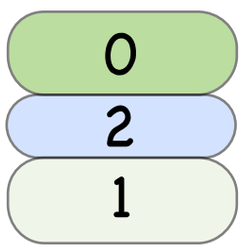
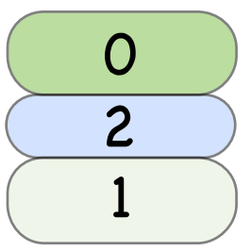
Creadora	• Vacía: Retorna una pila vacía.
Observadora	• Vacia?: Toma una pila y dice si esta está vacía o no.
Modificadora	• Apilar: pone un elemento dado en el tope de la pila, conservando los demás
Modificadora	• Desapilar: saca un elemento del tope y se conservan los demás.
Observadora	• Tope: devuelve el elemento en el tope, la colección de elementos no sufre modificaciones.

Especificando TADs

Una propiedad básica de las pilas es la siguiente:



Es decir la siguiente "ecuación" se cumple:

desapilar(apilar( , 7)) = 

Especificando TADs

Para especificar un TAD describimos las sus propiedades por medio de axiomas.

principales propiedades de las pilas

interface del TAD

$$vacia :\rightarrow Pila$$
$$apilar : Pila \times Elem \rightarrow Pila$$
$$desapilar : Pila \rightarrow Pila$$
$$tope : Pila \rightarrow Elem$$
$$vacia? : Pila \rightarrow Boolean$$

$$desapilar(vacia()) = error$$
$$desapilar(apilar(S, i)) = S$$
$$vacia?(vacia()) = true$$
$$vacia?(apilar(S, i)) = false$$
$$top(vacia()) = error$$
$$top(apilar(S, i)) = i$$

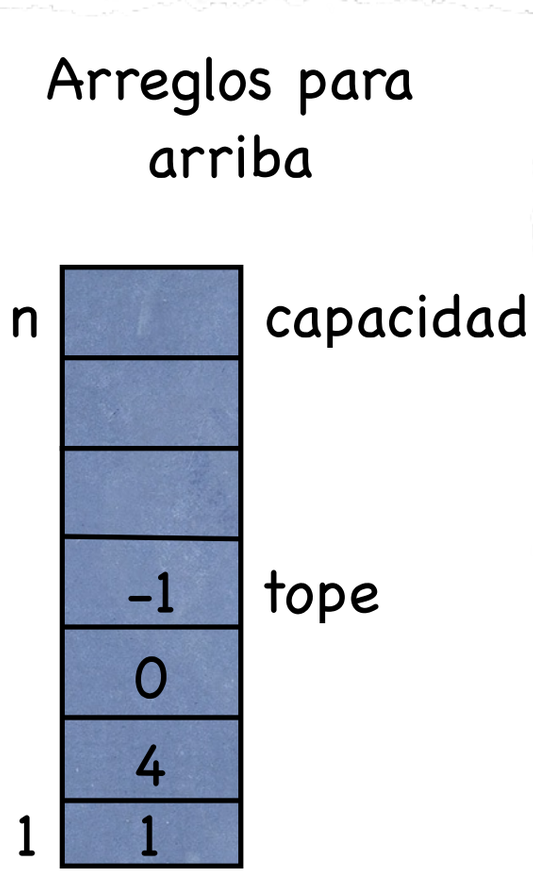
La noción de pila se define en base a las propiedades de sus operaciones

La especificación por axiomas es útil para investigar las propiedades algebraicas del TAD.

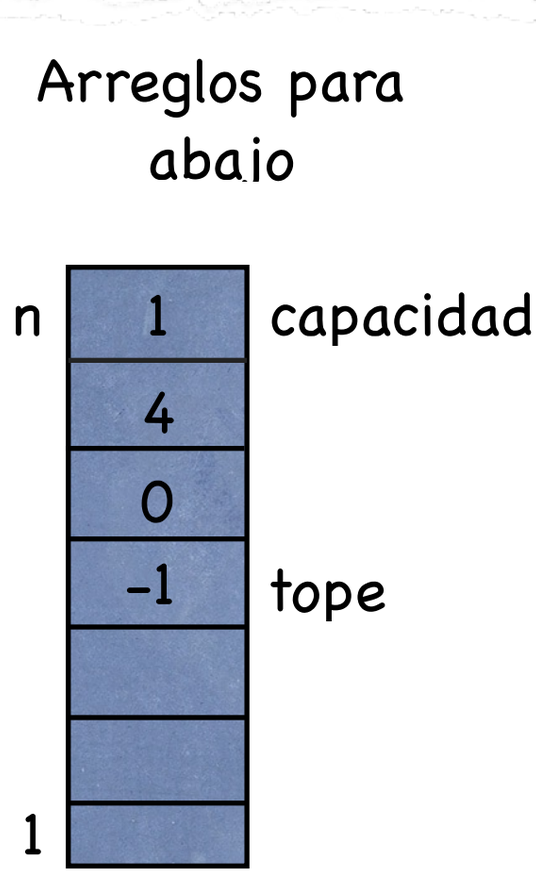
Implementando TADs

Un TAD puede tener diversas implementaciones, por ejemplo, en el caso de pila:

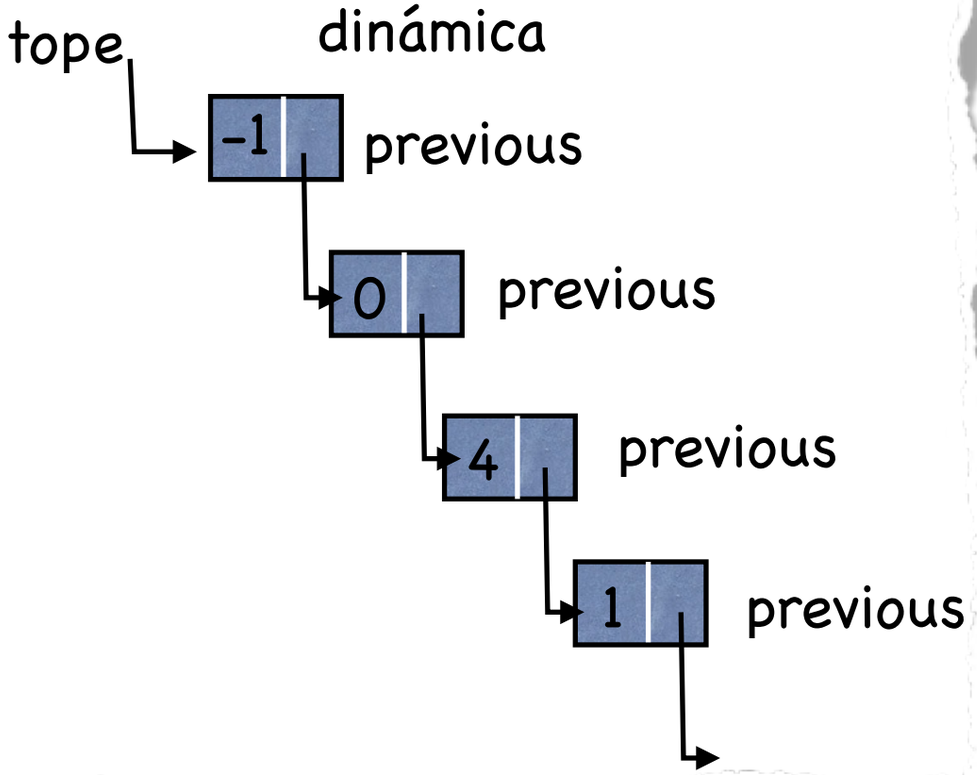
Arreglos para arriba



Arreglos para abajo



Con memoria dinámica



Todas implementan la misma pila!

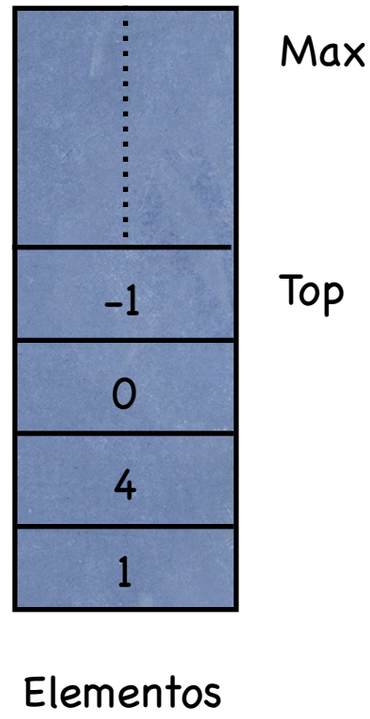
Implementación en Pascal

Podemos usar Pascal para implementar las pilas:

```
CONST
  Max = 100;

TYPE
    Pila = RECORD
      Elementos : ARRAY [1..Max] OF elemType;
      Top : 0 .. Max
END;
```

Implementación utilizando arreglos



Cada implementación tiene sus pros y sus contras

Implementando ADTs en Pascal

Para las operaciones proveemos los procedimientos y funciones necesarias, con pre y post-condiciones:

```
function vacia?(p:Pila):Boolean;
(*Pre: true*)
vacia? := (p.top = 0)
(*Post:  true ssi p es una sequencia vacia *)
END;
```

no tiene precondición

La pila es vacía

```
procedure desapilar(var p:Pila);
(*Pre: p.top > 0*)
p.top := p.top - 1 ;
(*Post: se saca el ultimo elemento de la pila*)
END;
```

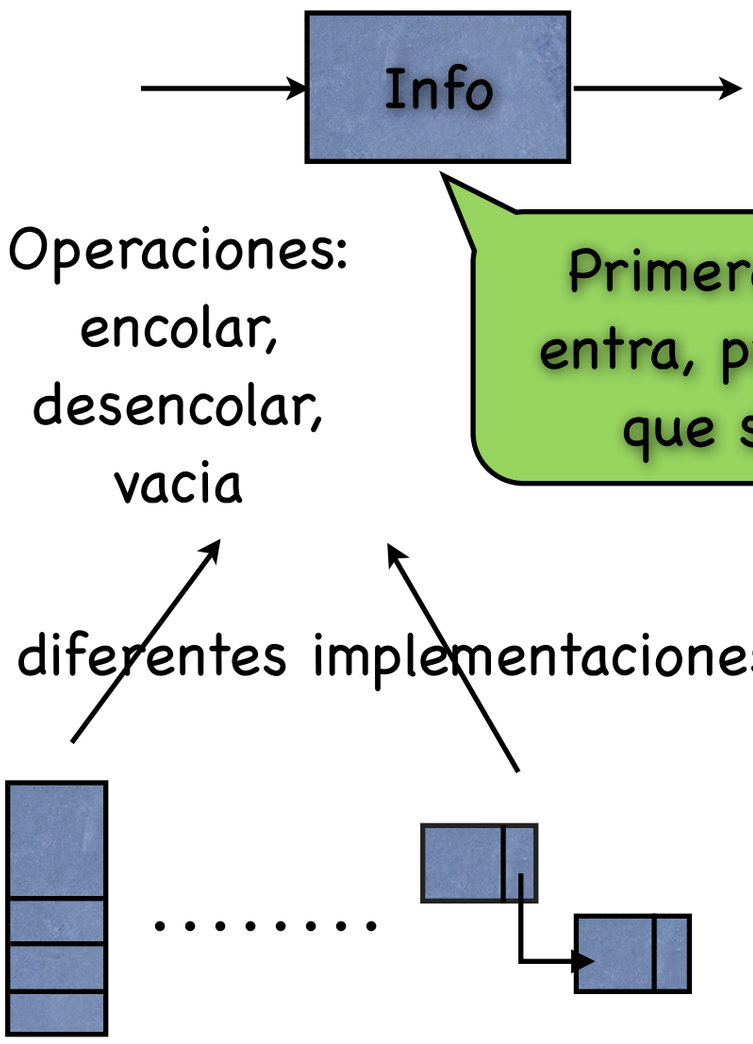
para que funcione correctamente debe haber espacio

Las pre y las post deben ser coherentes con los axiomas

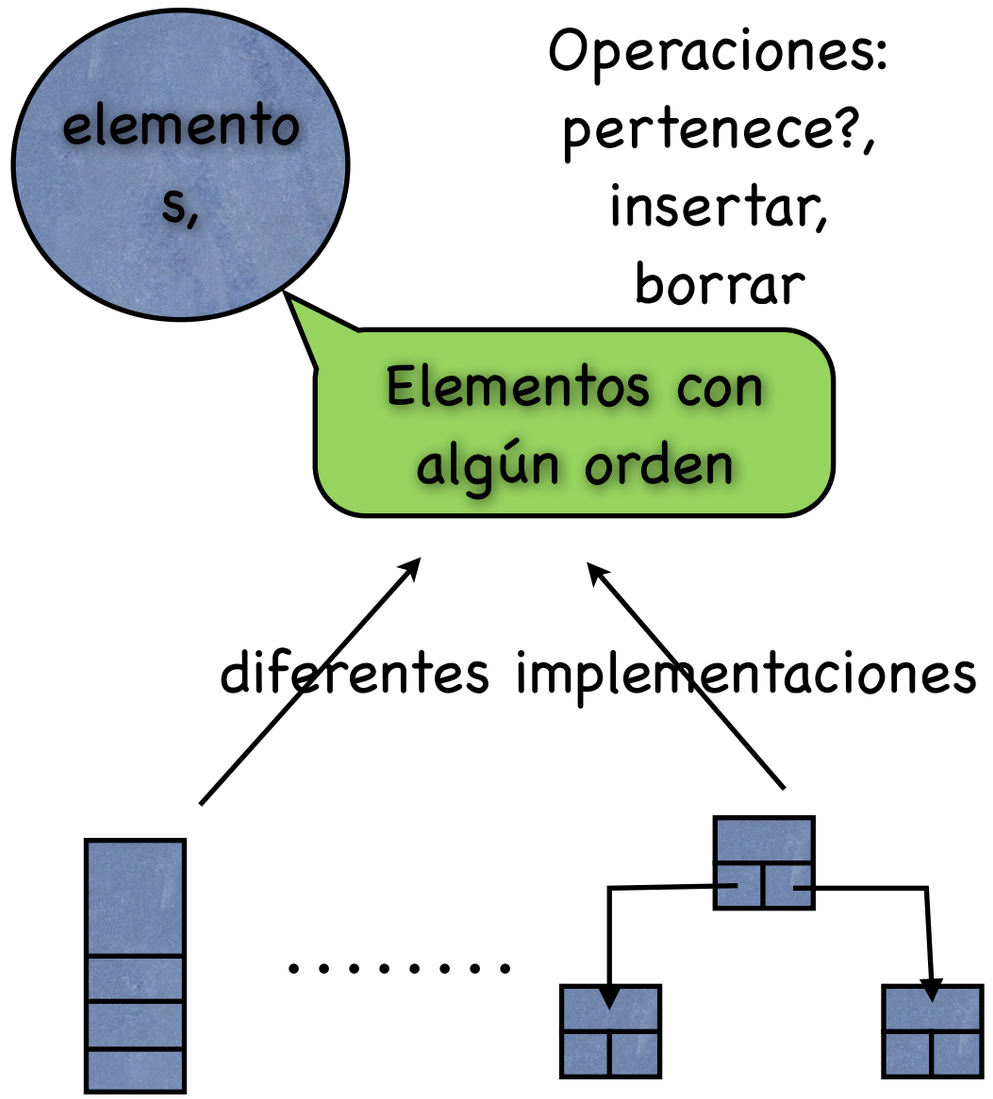
Otros TADs

Existen muchos TADs muy usados en programación, algunos ejemplos:

Colas



Diccionarios



Comentarios Finales

- Los lenguajes de programación más modernos proveen herramientas para implementar TADs, por ej: JAVA con interfaces y clases.

- Los TADs están fuertemente relacionados con conceptos importantes de Ing. de Software: Modularización, Ocultamiento de Información y Encapsulamiento.

POO

- Permite un mejor ocultamiento de información.
- Provee una mejor encapsulación de los modulos.
- El manejo de memoria dinámica es más transparente.
- Los programas son más estructurados.

Bibliografía

- A.Aho, J.Ullman, J.Hopcroft. Data Structures and Algorithms.
- B. Liskov, S. Zilles. Programming with Abstract Data Types.
- B. Meyer. Object Oriented Software Construction.
- W.Collins, T.McMillans. Implementing TADs in Turbo Pascal.